My goal was to build a functional cyberbullying detection model, but I also wanted to create a framework that could be improved over time. You'll see that reflected not just in the model itself, but in the tools I built around it.

Part 1

So, if you look at the Model_dev.ipynb file, you'll see I started with a Logistic Regression classifier that classifies each post as either bullying or non-bullying. It's a pretty straightforward choice, but I think it's effective. My thinking was to first establish a solid, interpretable baseline before trying anything more complex. The architecture is linear; it learns to associate certain words with a higher probability of bullying.

There are a couple of parameters I chose specifically that I think are crucial:

- **class_weight='balanced':** This was a must-have for me. The special highlight of this data is that it's always imbalanced. You have way more normal text than actual bullying. This parameter was my way of forcing the model to pay close attention to the bullying examples and adjust the weight, so it couldn't just get a high accuracy score by ignoring the minority class.

- **solver='liblinear':** This is the optimization algorithm I used. It's a reliable engine for smaller datasets like this one and handles the task efficiently.

- **Annotation Tool**: Annotators label posts (as bullying or self-harm or not) in a Streamlit interface, and their annotations help to assess the model's accuracy in labeling the posts and measures the inter-annotator agreement using Cohen's Kappa.

Data Processing

I started by pulling the data from a url provided from the challenge and parsing it line by line to separate the text from the labels.

Now, I have to be honest, my way of cleaning it is quite aggressive, and looking back, not the most efficient. I have it set to convert everything to lowercase, and then strip out all URLs, punctuation, emojis, and numbers. On one hand, it creates a very clean vocabulary for the model. On the other hand, I recognize that I'm losing a lot of context. So much of the nuance in online harassment is in the punctuation or the emojis used, and stripping them out is a tradeoff I made for simplicity at the start.

To actually create the features, I used a TfidfVectorizer. For me, this is where the text starts to speak a language the machine can understand. Instead of just counting words, TF-IDF creates a score for each word based on its frequency in a single post versus its rarity across all posts.

This gives a lot more weight to significant, potentially predictive words. This process creates a unique numerical fingerprint for every post, and that's what I fed into the model.

Model Training Method

The training process itself is pretty standard. I used train_test_split to set aside 20% of the data for testing, so the model would be evaluated on data it has never seen before. I set a random_state so that if I run it again, I get the exact same split, which is important for reproducing the results.

I then trained the model on the other 80% of the data using the .fit() method. This is where the model learns the patterns from the TF-IDF fingerprints and their corresponding labels. After it was trained, I used the .predict() method on the 20% test set to see how well it performed.

The training methodology involves 4 key steps mainly:

1. Data Collection and Processing: Clean raw posts and process the text.
2. Model Selection:: Use of logic regression to classify post.
3. Model Evaluation
4. Inter-Annotator Agreement

Model Performance and Analysis

Strengths: I think the model's main strength is its use of class_weight='balanced', which makes it much better at actually finding instances of bullying. It's a simple, efficient, and interpretable baseline, which is what I wanted in the beginning.

Weaknesses: As I mentioned earlier, the aggressive cleaning is a definite weakness. The model is limited because it can't understand the deictic nature of language. It is just looking at words. Sarcasm or complex bullying would go right over its head.

Part 2

Validation, Bias Mitigation, and Research Design
For validation, the focus is not just pure accuracy but also fairness and demonstrable real-world impact.

- **Algorithmic Bias Mitigation:** When validating a system like this, we have to pay attention to algorithmic bias. To address this head-on, the analyze_algorithm_bias function systematically evaluates model performance across different demographic

subgroups. This ensures we are actively searching for and can address any significant performance gaps.

- **A/B/C Test Framework:** To understand the system's true effectiveness, I designed a randomized A/B/C test in the design_validation_study function. Group A receives intervention messages immediately after a flagged post, Group B receives no intervention messages, and Group C receives intervention messages for a flagged post after 24 hours.

Intervention and Performance Metrics
The system's ultimate success is defined by its ability to create positive outcomes for users. The intervention_design.py file details how this is designed and measured.

- **Intervention Content:** My approach with the intervention design was rooted in a core belief: the system must aim to support, not to punish. The message templates are intentionally non-judgmental and supportive, designed to de-escalate situations. For critical cases like self-harm risk, the system's only role is to connect a user with professional help resources as quickly as possible.

- **Effectiveness Metrics:** The measure_intervention_effectiveness function outlines a practical set of metrics.

- **Short-Term Metrics:** We look for immediate behavioral changes, such as the "post_change_rate" (users editing or deleting a post after an intervention) and measurable shifts in post sentiment.

- **Long-Term Metrics:** While upholding strict privacy standards through data hashing and time-limited analysis, the plan is to compare re-offense rates to the control group to assess lasting impact.

Final Thoughts and Where I'd Go From Here

If i had more time, I would probably have switched to a pre-trained Transformer like BERT for nuance. BERT understands words in both directions, so it captures context. It knows that "killing it" is different from "killing someone."

Finally, I’d use my annotation app to create an "active learning" loop. The model could flag the posts it's most unsure about, and I could then label them. This would be a much more powerful and efficient way to make the model smarter over time. For me, that's the real path forward: a partnership between the human and the machine to build a better, more insightful tool.